Ada* における汎用体の実行プログラム共用化方式†

小泉 忍††

新しい計算機言語 Ada に特有の汎用体は，副プログラムおよびパッケージの雛型で，他言語のマクロ機能に相当するが，その実現方法が問題となっている．汎用体の具現は，ソースプログラム上において汎用体の仮パラメータを実パラメータで置換しコンパイルすることにより実現可能であるが，本論文では，汎用体本体を独立にコンパイルして単一の実行プログラムを生成し，汎用体の複数の具現に対してそれを共用する新方式を提案する．そのために，まず，汎用体のパラメータ，特に汎用仮型により汎用体実行プログラムが変化する範囲を明確にし，汎用体実行プログラムの共用可能性を示す．次に，ここで提案する汎用副プログラムに対するオブジェクトプログラムの概要を具体例を挙げて解説し，汎用パッケージへの適用可能性を論ずる．本論文で提案する方式では，実行プログラムを共有するためのオーバヘッドが存在するが，ソースプログラム上でパラメータを置換し展開する方式に比べ，具現に対するコンパイルが速い，汎用体の変更時に具現プログラムの再コンパイルが不要である等，プログラム開発時に有利な特徴を持つ．

1. まえがき

1983年2月に米国国家規格局 ANSI** 最終仕様が確定した新しい言語である Ada[1]は，装置組み込み型計算機用のシステムプログラムの記述を目的として，米国国防省 DoD***において提唱・開発された言語である．

Ada には，抽象データ型の概念を反映するパッケージ，システムプログラム記述のための並列処理，装置制御用として必要かつ十分な演算精度指定等，数多くの新機能が盛り込まれている．

Ada コンパイラの開発に当たっては，従来言語にない Ada 特有の機能の実現方式が問題となるが，その一つに汎用体 (generic unit) の問題が挙げられる．

汎用体とは，その内部で使用する副プログラムやデータ型をパラメータとして，複数の副プログラムやパッケージを作り出す際の雛型であり，高級言語におけるマクロ機能を提供するものである．一般にマクロ機能とは，ソースプログラムのレベルで仮パラメータを実パラメータに置換し展開することを言うが，汎用体についても同様の処理で機能が実現できる****．

汎用体の機能をソースプログラム上でマクロ展開することは，簡単なワードプロセッサ程度の処理で実現できるが，プログラム開発の効率の面で，具現時におけるコンパイル量が多い，汎用体本体の変更に伴う再コンパイル量が多い等の不都合が生じる．

これに対し，汎用体の単一実行プログラムを作成し，汎用体の使用時にそれを共用する方法が考えられ，それは特にプログラム開発時に有利である．汎用体の実行プログラムを共用する方式については文献2), 3) に言及があるが，具体的な手法は示されていない．また，Fortran, PL/1 等では，副プログラム名をパラメータとすることができるが，データ型をパラメータとする機能はない．

本論文では，特にデータ型をパラメータとする場合を中心に，オブジェクトプログラムの形式を具体的に示し，汎用体の実行プログラムを各使用時に共用する方式を提案する．

2. 汎用体の実現における問題点

Ada における汎用パッケージの代表例の一つに，標準入出力パッケージがある．

Ada には，言語の既定の型から，それと等価な構成を持つ別の型をユーザが定義する派生型 (derived type) の機能があるが，元の既定の型と派生した型とを厳密に区別するため，既定の型に関するパッケージや副プログラムを，そのまま派生型に適用することができない*．そこで Ada では，標準入出力を汎用パッケージとして準備し，実際に入出力を行う型それぞれについて，ユーザが具現 (instantiation) を行うことにより，それぞれの入出力パッケージを作成する構成と

† A Method for Sharing Executable Programs Derived from Generic Units in Ada by SHINOBU KOIZUMI (Systems Development Laboratory, Hitachi Ltd.).

†† (株)日立製作所システム開発研究所第2部

* Ada は米国政府の開発した言語であり，米国政府 AJPO (Ada Joint Program Office) の米国における登録商標である．

** American National Standard Institute

*** Department of Defence

**** 例えば ALS Ada コンパイラ[5]では，ソースプログラムに近い高級中間語上でパラメータ置換を行い汎用体の具現を実現している．

* ただしパッケージで宣言された型に対する操作副プログラムの一部は，型の派生に伴い暗黙に宣言されるものがある．

なっている.

概念的には，汎用体の仮パラメータを実パラメータで置き換えた新しいソースプログラムを仮定することにより，汎用体の具現を達成する．しかし，汎用体の具現をソースプログラム上のパラメータの置換で行うことには，以下に示すような問題がある．

(1) ユーザが定義した型は，計算機内部では，その計算機に固有の型から構成される．例えば，実際の入出力パッケージの機械語レベルのプログラムの内容は，計算機固有の型に対応するものに限られる．したがって，このような入出力パッケージの具現があるごとに，ソースプログラム上でパラメータの置換を行い，コード生成を繰り返すのは無駄である．

(2) 汎用体の仕様部 (generic specification) を変えずに，汎用体内部の処理を変更した場合，ソースプログラム上でパラメータの置換を行う方式では，その汎用体を使用している他のコンパイル単位のうち，汎用体の変更以前にコンパイルを終了したものについては，すべて再コンパイルする必要がある．

これは，テスト・デバッグ時におけるプログラム開発の効率を損なうばかりでなく，その汎用体を使用している他のコンパイル単位を完全に把握・管理する必要があり，開発支援系の負担が大きくなると予想される．

(3) 入出力パッケージのようにシステムが提供するプログラムの場合，一般にはアセンブリ言語，あるいは Ada 以外の低レベルのシステム記述言語で書く場合が多いと思われる．そのようなプログラムをオブジェクトプログラム，あるいは機械語レベルのみで提供する場合は，ソースプログラム上でパラメータを置換する方式は不可能である．

また，ソースプログラムレベルのパラメータ置換を実現するために，システムが提供するプログラムを Ada で記述することが考えられる．しかし一般には，そのようなプログラムは計算機のハードウェアに依存する部分が多く，Ada の機能の中でも内部表現節 (representation clause) を多用することが予想され，汎用体のような抽象化された枠組の中で効率の良いプログラムを書くことができるかどうか疑問である．

以上のような問題点の解消を図るのが，本論文で提案する方式の目的である．なお，ここで提案する方式と他の方式との比較については，5章で述べる．

3. 汎用体実行プログラムの共用可能性

2章で示した問題点(1)および(2)は，汎用体本体 (generic body) のコピーを具現ごとにコンパイルすることにより生じている．そこで，汎用体本体に対して単一のオブジェクトプログラムを考え，その汎用体のそれぞれの具現に対して，そのオブジェクトプログラムを共用することを考える．以下そのような方式を，汎用体実行プログラム共用方式と呼ぶ．

汎用体実行プログラム共用方式では，汎用体のオブジェクトプログラムにおいて各具現に共用できる部分と共用できない部分，すなわち汎用体仕様部で与えられた汎用パラメータに関係し汎用体の各具現で変化する部分と，具現に対しては汎用体として不変である部分とを明確に分けることが基本となる．

汎用仮パラメータ (generic formal parameter) には，汎用仮算体 (generic formal object)，汎用仮副プログラム，汎用仮型 (generic formal type) の3種類があるが，以下では，それぞれの汎用仮パラメータに対する実パラメータが汎用体本体に及ぼす影響について考察する．

(1) 汎用仮算体

汎用仮算体は汎用体に対する大域的算体を意味し汎用体の具現時には具現された環境における大域的実算体と結合する．汎用仮算体とそれに対する実算体との関係は，一般の副プログラムの仮パラメータと実パラメータの関係によく似ている．一般の副プログラムのパラメータの場合，概念的には，副プログラム中では実パラメータの値のコピーに対して操作を行うのが Ada の原則である．それに対し汎用仮算体に対する実算体の場合，概念としてはソースプログラムレベルでのパラメータの置換えを行うので，具現された汎用体ではパラメータである実算体を直接操作する点が一般の副プログラムのパラメータと異なる．したがって，汎用仮算体に対する実算体は完全に汎用体の外部に存在し，一般の副プログラムのように仮パラメータのための記憶領域を考える必要がない．汎用仮算体が汎用仮型である場合を除き，汎用仮算体と実算体の結合により変化するのは，その実算体の存在場所だけである．よって，汎用仮算体と実算体の結合方法としては，いわゆる参照（アドレス）渡しを用いることで汎用体実行プロ

グラムの共用を図ることができる．汎用仮算体が汎用仮型である場合の問題は後述する．

（2） 汎用仮副プログラム

汎用仮副プログラムは汎用体に対する外部副プログラムを意味し，汎用体の具現時に具体的な副プログラムと結合する．Ada以外の言語では，副プログラムのパラメータとして副プログラム名を渡せるものがあり，Adaの汎用仮副プログラムの機能はそれとよく似ている．汎用仮副プログラムに対する実副プログラムは常に汎用体の外部にあり，具現により変化するのは，実副プログラムの存在場所と，汎用仮副プログラムの副プログラムとしての仮パラメータが汎用仮型となっていれば，その実プログラムの仮パラメータの型である．したがって基本的には，具現における実副プログラムの入口アドレスをパラメータとして渡す機構を用いれば汎用体の実行プログラムの共用を図ることができる．また，副プログラムの仮パラメータは，その副プログラムの局所変数とほぼ同様の取扱いとなるので，汎用仮副プログラムの副プログラムとしての仮パラメータが汎用仮型である場合の問題については次項を参照されたい．

（3） 汎 用 仮 型

汎用仮型は，汎用体の具現において算体やパラメータの型を決定するために用いる．

一般に型とは，その型に属する値の「集合」と，その集合の要素である値に対する「操作」の二つにより特徴付けられる．Adaでは，値の形式と値の範囲により「集合」を規定することで型を定義（宣言）する．一方「操作」については，値の形式の種類に従って，宣言をせずに使える「基本操作（basic operation）」が言語仕様により定められている．汎用仮型は7種類あるが，それぞれの種類に適用される基本操作も言語仕様で規定されている．以下では基本操作に関する問題と値の形式に関する問題の2点を示し，それぞれの解決方法を述べる．

a．基本操作に関する問題

汎用体本体の実行部の手続きを記述する場合，汎用仮型の算体については，その仮型に関する基本操作と，その仮型に関する副プログラム，および，汎用仮副プログラムを用いる．基本操作は，本来，型に固有のものであるから，汎用仮型に対する具現時の実型が異なれば，その操作の具体的な内容は異なったものになるはずである．すなわち，汎用仮型が汎用体の不変性に与える影響の第一点は，型に付随する基本操作が具現により暗黙のうちに決定される点である．

この問題に対しては，それぞれの型に関する基本操作自体を副プログラム化し，汎用仮副プログラムと同様の取扱いをすることが考えられる．すなわち，汎用体本体中で用いられている汎用仮型基本操作が汎用仮副プログラムとして宣言されていると見なし，具現時には対応する実型の基本操作を実副プログラムとして結合するという方法である．汎用仮型に対する基本操作の種類は，実型の基本操作の種類のサブセットであり，実型の基本操作を副プログラム（実行時ルーチン）として準備することは可能である．よって，基本操作が汎用仮型に対する実型により変化する点を，上記方法により回避することができる．また，汎用体中に汎用仮型を用いた内部副プログラムがあったとすると，この内部副プログラム中の汎用仮型算体に対する基本操作はすべて基本操作に還元できる．したがって，前記のように汎用仮型に関する基本操作を外付けにすることは，汎用体の共用実行プログラム中では，汎用仮型算体の具体的操作をまったく行わず，それらに対するデータフローの制御のみを行うことを意味する．

なお，Adaでは基本操作を行っている時に例外（exception）が発生することがある*．Adaでは副プログラム中に例外処理部を置くことができるが，例外処理部のない副プログラム中で発生した例外は，その副プログラムを呼び出した副プログラムに伝播し，呼び出し側で処理を行う．汎用仮型に対する実型の算体に対し基本操作を行っている際に例外が発生した場合，副プログラム化した基本操作ルーチンに例外処理部を置かなければ，呼び出し側，すなわち，汎用体の共用実行プログラムに例外を伝播し，そこで例外処理を行う．このように，基本操作を副プログラム化しても，基本操作が副プログラムとして分離されていない場合と同様の例外処理が行える．

b．型の値の形式の問題

汎用仮型に対する実型算体の具体的操作を，汎用体共用プログラム中で直接に行わないとすると，その実型の値の具体的な表現形式に関する情報は，基本的には共用実行プログラムには不要で

* 例えば代入時における制約（constraint）エラー

```
 1  generic
 2     type T is private;
 3  procedure SWAP( X, Y: in out T );
 4
 5  procedure SWAP( X, Y: in out T ) is
 6     TEMP: T;
 7  begin
 8     TEMP := X;
 9     X    := Y;
10     Y := TEMP;
11  end;
```

図 1 汎用副プログラムの例
Fig. 1 Example of a generic subprogram.

ある．しかしこれに関連して次のような問題が生じる．すなわち，汎用仮型算体に必要な記憶域の大きさは対応する実型によって決まるため，汎用体に局所的な汎用仮型算体を直接に局所記憶領域に割付けることは困難である．これが，汎用仮型が汎用体の不変性に与える影響の第2点である．

このように大きさが事前に決まらない算体に対しては，算体自身の領域は動的変数領域にとり，局所変数領域の固定部にはその算体の存在場所を示すポインタを置き，実際に算体を参照・操作する時にはそのポインタを介して間接的にアクセスする方法をとることが考えられる．この場合，汎用体の共用実行プログラムには，その入口で局所変数の領域を確保するために，汎用仮型に対する実型算体に必要な領域の大きさの情報だけは必要である．すなわち，汎用体の具現時には，汎用体共用実行プログラムに対して，実型算体の大きさをパラメータとして渡し，前記方法をとることで汎用仮型が汎用体の不変性に与える影響の第2点も回避することができる．

以上の点を考慮して汎用体のオブジェクトプログラムを作成すれば，実行時に汎用仮パラメータに対応する実パラメータを渡すことで汎用体の具現を実現することができ，汎用体のオブジェクトプログラムをその汎用体のすべての具現で共用することが可能となる．

オフセット	
0	仮型Tに対する実型の物理サイズ
4	実型の代入操作ルーチン入り口
8	・
・	・
・	・

図 2 汎用副プログラム SWAP のリンクテーブルの形式
Fig. 2 Link table format for the generic subprogram SWAP.

```
SWAP  EQU *                  -- 汎用ＳＷＡＰプログラム
      PROLOG                 -- スタックフレームの確保
      L   Ra,0(PL)           -- リンクテーブルアドレスを獲得
      ST  SP,i(FP)           -- ＴＥＭＰのアドレスを設定
      A   SP,0(Ra)           -- ＴＥＭＰの領域を確保（スタックの最後）
*
      ST  PL,j(FP)           -- パラメタリストアドレスの退避
      MVC k  (4,FP),4(PL)    -- パラメタリストにＸのアドレスを設定
      MVC k+4(4,FP),i(FP)    -- パラメタリストにＴＥＭＰのアドレスを設定
      LA  PL,k(FP)           -- ＰＬにパラメタリストアドレスを設定
      L   15,4(Ra)           -- ＧＲ１５に代入操作ルーチンアドレスを設定
      BALR 14,15             -- サブルーチンコール（ＴＥＭＰ：＝Ｘ；）
*
      L   PL,j(FP)           -- パラメタリストアドレスの回復
      MVC k  (4,FP),8(PL)    -- パラメタリストにＹのアドレスを設定
      MVC k+4(4,FP),4(PL)    -- パラメタリストにＸのアドレスを設定
      LA  PL,k(FP)           -- ＰＬにパラメタリストアドレスを設定
      L   15,4(Ra)           -- ＧＲ１５に代入操作ルーチンアドレスを設定
      BALR 14,15             -- サブルーチンコール（Ｘ：＝Ｙ；）
*
      L   PL,j(FP)           -- パラメタリストアドレスの回復
      MVC k  (4,FP),i(FP)    -- パラメタリストにＴＥＭＰのアドレスを設定
      MVC k+4(4,FP),8(PL)    -- パラメタリストにＹのアドレスを設定
      LA  PL,k(FP)           -- ＰＬにパラメタリストアドレスを設定
      L   15,4(Ra)           -- ＧＲ１５に代入操作ルーチンアドレスを設定
      BALR 14,15             -- サブルーチンコール（Ｙ：＝ＴＥＭＰ；）
*
      EPILOG                 -- スタックフレームの解放
      END
```

図 3 汎用副プログラム SWAP のオブジェクトプログラム
Fig. 3 Object program for the generic subprogram SWAP.

4. 汎用体実行プログラム共用方式の概要

4.1 汎用副プログラムの場合

前章における考察に基づく汎用副プログラムの実現方式の概要を，例を挙げ以下に示す．

(1) 汎用体のコンパイル時処理

図1に示した汎用副プログラム SWAP は，汎用仮型Tのパラメータ X, Y で指示された二つの算体の値を入れ替えるものである．Ada では，外部とのインタフェースを記述する仕様部と，内部処理を記述する本体とを分離するが，それぞれの機能分担に基づき，次の処理を行う．

a．汎用体仕様部

汎用体仕様部に対しては，汎用仮パラメータの種類に対応して，次の要素から成るリンクテーブルの雛型を作成する．

(i) 汎用仮算体に対応する実算体へのポインタ

(ii) 汎用仮副プログラムに対応する実副プログラムの入口点アドレス

(iii) 汎用仮型に対する実型算体の大きさ，および実型の基本操作ルーチンの入口点アドレス

図2に汎用副プログラム SWAP のリンクテーブルの形式を示す．(この場合 (iii) 相当のみである．)

b．汎用体本体

汎用体本体に対しては，次の点を特徴とするオブジェクトプログラムを生成する．

(i) 汎用仮型に対する実型の算体に対し，その大きさを前記リンクテーブルから得て，算体の領域を動的算体領域上に確保し，そこへのポインタを局所算体用固定領域上に設定することを含むプロローグ処理を行う．

(ii) それぞれの汎用仮パラメータに対応する実算体，実副プログラム，実型の基本操作等については，前記リンクテーブルを参照することによる間接アクセス方式とする．

(iii) 汎用仮型に対する実型算体の操作は，基本操作を含め，すべて，その算体へのポインタをパラメータとした副プログラム呼び出しにより行う．

汎用副プログラム SWAP の本体に対するオブジェクトプログラムの例を**図3**に示す．なお，以下の各図におけるオブジェクトプログラムの記述は，HITAC M シリーズ用アセンブラ言語に準拠している．また図中の FP, SP, PL は，それぞれスタックフレーム先頭アドレス，スタック最終アドレス，パラメタリスト先頭アドレスを示すレジスタである．

```
procedure ...
        .
        .
   procedure INT_SWAP is new SWAP( T => INTEGER );
   procedure REAL_SWAP is new SWAP( T => FLOAT );
   A, B: INTEGER;
   C, D:FLOAT;
        .
        .
begin
        .
        .
   INT_SWAP( A, B);
        .
        .
   REAL_SWAP( C, D );
        .
        .
end;
```

図4 汎用副プログラム SWAP の使用例
Fig. 4 Example of the use of the generic subprogram SWAP.

(2) 汎用体の具現，および，使用に対する処理

汎用副プログラム SWAP を INTEGER 型に対して具現した INT_SWAP と，FLOAT 型に対して具現した REAL_SWAP，および，それぞれの呼び出しの例を，**図4**に示す．

a．汎用体の具現

汎用体の具現においては，汎用仮パラメータに対応するそれぞれの実パラメータにより，実際のリンクテーブルを作成する．

図5は，図4のプログラムに対するオブジェクトプログラムの例である．ここで，INTSWAPTB および REALSWAPTB は，それぞれ INT_SWAP, REAL_SWAP に対するリンクテーブルの先頭アドレスである．

また，**図6**は INTEGER 型に対する代入操作ルーチン INTASGN，**図7**は FLOAT 型に対する代入操作ルーチン REALASGN のオブジェクトプログラムの例である．

b．具現副プログラムの呼び出し

具現した副プログラムの呼び出しは，通常の副プログラムの実パラメータのほかに，前記 a で作

成したリンクテーブルの先頭アドレスを第1パラメータとして，汎用体本体に対する実行プログラムを呼ぶことにより実現する．

以上，例で示したように，この方法によれば，汎用副プログラムの実行プログラムを，各具現に対して共用し，実行することが可能である．

(3) 汎用体共用実行プログラムのリンケージ

レコード型・配列型・および表現節によりユーザが型の内部表現を規定した場合を除き，型の計算機内部における表現は，システムの既定のものとなる．したがって，システムの既定の内部表現をもつ型の基本操作ルーチンは，他の実行時ルーチンと同様，システムが事前に準備し，ライブラリ化する．またレコード型・配列型・および表現節がある型の場合は，各具現に対応して，それぞれ必要な基本操作ルーチンを生成することが必要である．

```
INTSWAPTB  EQU *            --  INT_SWAP用リンクテーブル
           DC  4            --  INTEGER型サイズ 4バイト
           DC  V(INTASGN)   --  INTEGER型代入操作ルーチンアドレス
               .
               .
REALSWAPTB EQU *            --  REAL_SWAP用リンクテーブル
           DC  8            --  FLOAT型サイズ 8バイト
           DC  V(REALASGN)  --  FLOAT型代入操作ルーチンアドレス
               .
               .
A          DS  4            --  変数Aの領域
B          DS  4            --  変数Bの領域
C          DS  8            --  変数Cの領域
D          DS  8            --  変数Dの領域
               .
               .
    MVC k  (4,FP),=A(INTSWAPTB)  --  INTSWAPTBを用いた
    MVC k+4(4,FP),=A(A)          --  汎用SWAPの実行
    MVC k+8(4,FP),=A(B)          --  (INT_SWAP(A, B);)
    LA  PL,k(FP)
    L   15,=V(SWAP)
    BALR 14,15
               .
               .
    MVC k  (4,FP),=A(REALSWAPTB) --  REALSWAPTBを用いた
    MVC k+4(4,FP),=A(C)          --  汎用SWAPの実行
    MVC k+8(4,FP),=A(D)          --  (REAL_SWAP(C, D);
    LA  PL,k(FP)
    L   15,=V(SWAP)
    BALR 14,15
               .
```

図5 汎用副プログラム SWAP の具現と使用に対するオブジェクトプログラムの例
Fig. 5 Example of the object program for the instantiation and the use of the generic subprogram SWAP.

図8は，汎用副プログラムを具現し，使用する際のオブジェクトプログラムの概観である．このように，汎用体の各具現時に作成するリンクテーブルは具現を行うプログラムの定数部に置く．

リンクテーブルの内容のうち，基本操作ルーチンの入口点アドレスを含む各アドレス定数は，通常の外部副プログラムや実行時ルーチンと同様，リンケージエディタにより最終的に決定する．

また，汎用体と，それを使用するプログラムとが異なったコンパイル単位にある場合があるが，これらのリンクは，汎用体共用実行プログラムの入口アドレスのみを決定すればよい．したがって，このリンクは，実行前にアドレスを決定し結合する静的リンクと，汎用体を使用するプログラムの実行時に汎用体共用実行プログラムをロードし結合する動的リンクの両方が可能である．静的リンクは，汎用体本体の変更が生じない場合に，また動的リンクは，汎用体本体の変更がある場合に有効な方法である．

```
INTASGN  EQU *              --  INTEGER型の代入操作
         PROLOG
         L   W1,0(PL)       --  代入元アドレスの設定
         L   W2,4(PL)       --  代入先アドレスの設定
         MVC 0(4,W2),0(W1)  --  転送
         EPILOG
         END
```

図6 INTEGER 型の代入操作ルーチン
Fig. 6 Assignment operation routine for type INTEGER.

```
REALASGN EQU *                -- FLOAT型の代入操作
         PROLOG
         L   W1,0(PL)         -- 代入元アドレスの設定
         L   W2,4(PL)         -- 代入先アドレスの設定
         MVC 0(8,W2),0(W1)    -- 転送
         EPILOG
         END
```

図 7 FLOAT 型の代入操作ルーチン
Fig. 7 Assignment operation routine for type FLOAT.

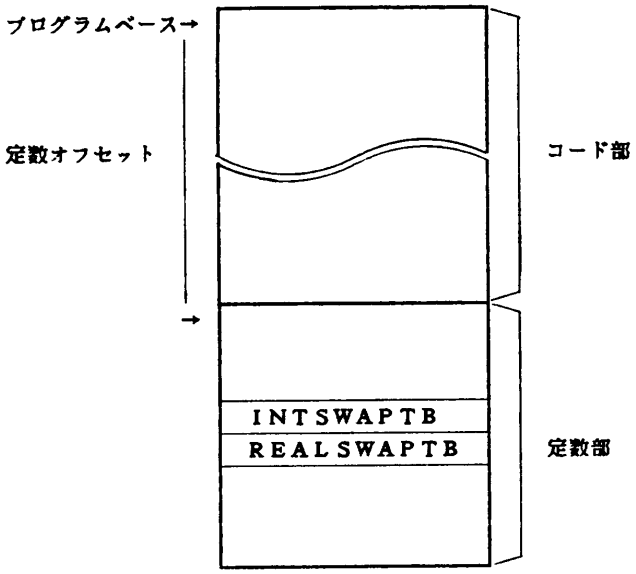

図 8 汎用体を使用するプログラムのメモリマップ
Fig. 8 Memory map for a program that uses a generic unit.

4.2 汎用パッケージへの拡張

Ada におけるパッケージには，データ（算体）の集合体としての側面と，操作（副プログラム）の集合体としての側面の二つの面がある．以下では，それぞれの側面から，汎用パッケージにおける実行プログラムの共用可能性について検討する．

（1） 操作の集合体としての汎用パッケージ

操作の集合体として汎用パッケージを見た場合，パッケージは，汎用副プログラムと一般の副プログラムから構成されていると見なせる．すなわち，汎用パッケージ中の副プログラムは，汎用パラメータが関係する限りにおいて汎用副プログラムの一種であり，汎用パッケージが具現される時，そのようなすべての副プログラムが具現されると考えられる．

パッケージに対する実行プログラムは，パッケージ初期化部分を除き，パッケージ中の副プログラムの実行プログラムから構成される．したがって，この場合，汎用パッケージに対する共用実行プログラムの基本は，汎用副プログラムに対するものと同一である．

（2） データの集合体としての汎用パッケージ

データの集合体としての側面における問題は，データの配置場所とそれへのアクセス方法である．

パッケージ宣言部の算体は，副プログラムの局所算体と異なり，その有効範囲がパッケージ外部にも及び，パッケージ自体の宣言と同じ有効範囲を持つ．すなわち，パッケージ宣言部の算体は，パッケージ自体と同じレベルの局所算体領域に置く．特にこれら算体が汎用仮型である場合は，汎用副プログラムの汎用仮型の算体と同様，算体本体を動的領域に置き，局所算体の固定領域にその算体へのポインタを設定し，間接アクセスを行う．

以上のようにして，汎用パッケージにおいても，実行プログラムの共用化が可能である．

5. 他方式との比較

ALS* Ada コンパイラ[5]では，汎用体のそれぞれの具現時に，高級中間語 Diana[4] 上で汎用仮パラメータ名を実パラメータ名に置換し，コード生成を行うことにより汎用体の具現を実現している．以下，ソースプログラム，あるいは高級中間語レベルでのパラメータ名置換による方式を，マクロ展開方式と呼ぶ．

以下では，本論文で提案した汎用体実行プログラム共用方式とマクロ展開方式について比較する．

（1） コンパイル時性能

第2章で述べたように，マクロ展開方式では，汎用体の具現それぞれについて汎用体全体のコード生成を行う，あるいは，汎用体本体の変更に伴い，その汎用体の具現すべてを再コンパイルする必要があり，全体としてコンパイル速度は低下する．それに対して汎用体実行プログラム共用方式では，そのようなことが不要であり，コンパイル速度が速い．すなわち，プログラム開発時の効率の点では，汎用体実行プログラム共用方式の方がマクロ展開方式より優れている．

（2） 実行時性能

汎用体実行プログラム共用方式では，汎用仮型に

* Ada Language System

関する基本操作がすべて副プログラムの形式となっているため，その呼び出しのオーバヘッドが存在する．基本操作の多くは，機械語の数命令で直接実現できると考えられ，汎用仮型を多用した場合，このオーバヘッドは無視できない．ただし，レコード型や配列型等の複合型をはじめとして，計算機の機種によっては浮動小数点型を含む特定の型の基本操作を実行時ルーチンにより実現する場合もあり，それらについては，呼び出しによるオーバヘッドはほぼ等しい．

また，汎用体実行プログラム共用方式では，汎用体本体のソースプログラムレベルで可能な大域的最適化を行うことはできる．しかし，機械語レベルで行う局所最適化については，マクロ展開方式に比べて行うことのできない部分があると考えられる．

さらに，目標となる計算機の機種によっては，汎用体実行プログラム共用方式で多用する間接アクセス方式を実現するのに，直接アクセスする場合に比べより多くの命令を要するものがある．また，1命令で間接アクセスができる場合でも，一般に直接アクセスする場合に比べ命令実行時間を多く要する．

以上の理由から，汎用体実行プログラム共用方式によるプログラムの実行速度は，マクロ展開方式に比べ低下する．

6. むすび

本論文では Ada における汎用体本体の実行プログラムを，汎用体のそれぞれの具現・使用において共用する方式を提案した．

本方式の基本は，抽象データ型の概念に基づき，汎用仮型と，それに対する基本操作に着目した点，すなわち，汎用仮型の基本操作を汎用仮型に伴ってパラメータ化し，汎用体の具現時に，汎用仮型に対する実型とともに，その基本操作を決定することにある．

本論文では，汎用仮型の基本操作を含めた汎用仮パラメータを実行プログラム上で実現する方法として，間接アドレス参照による結合方式の具体例を示した．

本方式によれば，ソースプログラム上で汎用仮パラメータを実パラメータに置換する方式に比べて，実行速度の点で劣るが，汎用体を使用する具現に対するコンパイル速度が速い点，また汎用体本体の内部処理を変更した場合に他のコンパイル単位に与える影響がない点等，プログラム開発の効率の点で優れている．

本論文では，汎用体本体の実行プログラムの共用可能性を検討するため，汎用副プログラムを中心として，計算機機種によらない一般論として汎用体プログラム共用方式の概要を述べたが，特定機種向に詳細化するのに大きな障害はない．特に，テーブル参照による間接アクセスを直接実行する間接アドレス命令を持つ計算機の場合，本論文で提案した汎用体実行プログラム共用方式の実現は容易である．

謝辞 本件につき，有益なご討論をいただいた立教大学 島内剛一教授に感謝いたします．

参考文献

1) U. S. Department of Defence: ANSI/MIL-STD-1815 A-1983 Ada Programming Language, p. 300, American National Standard Institute Inc. (1983).
2) 米田信夫編：プログラミング言語 Ada, *bit* 臨時増刊, pp. 1269-1495, 共立出版 (1981).
3) Bray, G.: Implementation Implecations of Ada Generics, *ACM Ada TEC-Ada Letters*, Vol. 3, No. 2, pp. 62-71 (1983).
4) Goos, G. and Wulf, Wm. A. (eds.): *Diana Reference Manual*, p. 140, Universitaet Karlsruhe, Karlsruhe (1981).
5) Simpson, R. T.: The ALS Ada Compiler Front End Architecture, *ACM Ada TEC Conference on Ada*, pp. 98-106 (1982).

(昭和59年8月 9 日受付)
(昭和61年4月17日採録)

小泉 忍（正会員）

昭和31年生．昭和55年東京工業大学工学部制御工学科卒業．昭和57年同大学院総合理工学研究科修士課程（システム科学専攻）修了．同年(株)日立製作所システム開発研究所入社．以来，現在に至るまで，主に汎用大型計算機を対象としたコンパイラの研究・開発に従事．コンピュータアーキテクチャ，プログラム変換等に興味を持つ．日本ソフトウェア科学会会員．